**MLOps**

MLOps refers to a methodology that is built on applying DevOps practices to machine learning workloads. Like DevOps, MLOps relies on a collaborative and streamlined approach to the machine learning development lifecycle where the intersection of people, process and technology optimize the end-to-end activities required to develop, build, and operate machine learning workloads.

* [**MLOps vs DevOps**](https://www.datasciencecentral.com/profiles/blogs/mlops-vs-devops-the-similarities-and-differences)

[MLOps and DevOps](https://www.datasciencecentral.com/profiles/blogs/mlops-vs-devops-the-similarities-and-differences) are similar when it comes to continuous integration of source control, unit testing, integration testing, and continuous delivery of the software module or the package.  However, in ML there are a few notable differences:

* **Continuous Integration**(CI) is no longer only about testing and validating code and components, but also testing and validating data, data schemas, and models.
* **Continuous Deployment** (CD) is no longer about a single software package or service, but a system (an ML training pipeline) that should automatically deploy another service (model prediction service) or roll back changes from a model.
* **Continuous Testing** (CT) is a new property, unique to ML systems, that’s concerned with automatically retraining and serving the models.

# **Need of MLOps**

As you move from running individual artificial intelligence and machine learning (AI/ML) projects to transform your business at scale, the discipline of ML Operations (MLOps) can help. MLOps accounts for the unique aspects of AI/ML projects in project management, CI/CD, and quality assurance, helping you to improve delivery time, reduce defects, and make data science work more productive.

MLOps is the discipline of integrating ML workloads into release management, CI/CD, and operations. It requires the integration of software development, operations, data engineering, and data science.

* **Challenges with MLOps**
* **Project management**

ML projects involve data scientists, a relatively new role, and one not often integrated into cross-functional teams. These new team members often speak a very different technical language than product owners and software engineers, compounding the usual problem of translating business requirements into technical requirements.

* **Communication and collaboration**

Building visibility on ML projects and enabling collaboration across different stakeholders such as data engineers, data scientists, ML engineers, and DevOps is becoming increasingly important to ensure successful outcomes.

* **Everything is code**
* Use of production data in development activities, longer experimentation lifecycles, dependencies on data pipelines, retraining deployment pipelines, and unique metrics in evaluating the performance of a model.
* Models often have a lifecycle independent of the applications and systems integrating with those models.
* The entire end-to-end system is reproducible through versioned code and artifacts. DevOps projects use Infrastructure-as-Code (IaC) and Configuration-as-Code (CaC) to build environments, and Pipelines-as-Code (PaC) to ensure consistent CI/CD patterns. The pipelines have to integrate with Big Data and ML training workflows. That often means that the pipeline is a combination of a traditional CI/CD tool and another workflow engine. There are important policy concerns for many ML projects, so the pipeline may also need to enforce those policies. Biased input data produces biased results, an increasing concern for business stakeholders.
* **CI/CD**
* In MLOps, the source data is a first-class input, along with source code. That’s why MLOps calls for versioning the source data and initiating pipeline runs when the source or inference data changes.
* Pipelines must also version the ML models, along with inputs and other outputs, in order to provide for traceability.
* Automated testing must include proper validation of the ML model during build phases and when the model is in production.
* Build phases may include model training and retraining, a time-consuming and resource-intensive process. Pipelines must be granular enough to only perform a full training cycle when the source data or ML code changes, not when related components change.
* Because machine learning code is typically a small part of an overall solution, a deployment pipeline may also incorporate the additional steps required to package a model for consumption as an API by other applications and systems.
* **Monitoring and logging**
* The feature engineering and model training phases needed to capture model training metrics as well as model experiments. Tuning an ML model requires manipulating the form of the input data as well as algorithm hyperparameters, and systematically capture those experiments. Experiment tracking helps data scientists work more effectively and gives a reproducible snapshot of their work.
* Deployed ML models require monitoring of the data passed to the model for inference, along with the standard endpoint stability and performance metrics. The monitoring system must also capture the quality of model output, as evaluated by an appropriate ML metric.
* **Benefits of MLOps**

It facilitates creation, automation and end-to-end management of ML workflows at scale, leading to faster production of ML models. In addition to managing code, SageMaker projects enable MLOps for model building, model deployment, and end-to-end ML workflows. You can run training jobs or SageMaker pipelines to build models in Amazon SageMaker Studio.

It also provides MLOps templates which provision the underlying resources needed for CI/CD capabilities. It also allows a trigger-based approach (like one based on code change) for model generation, unlike pipelines where the notebook cells have to be executed individually for the steps to be executed.

## How to implement MLOps

There are 3 ways to do this:

### **MLOps level 0 - Manual process**

This is typical for companies that are just starting out with ML. An entirely manual ML workflow and the data-scientist-driven process might be enough if your models are rarely changed or trained.

### **MLOps level 1 - ML pipeline automation**

The goal of MLOps level 1 is to perform continuous training (CT) of the model by automating the ML pipeline. This way, you achieve continuous delivery of model prediction service.

### **MLOps level 2 - CI/CD pipeline automation**

For a rapid and reliable update of pipelines in production, you need a robust automated CI/CD system. With this automated CI/CD system, your data scientists rapidly explore new ideas around feature engineering, model architecture, and hyperparameters. This is the one which is used nowadays, the other two are no longer in use.

* **Hands-on working**

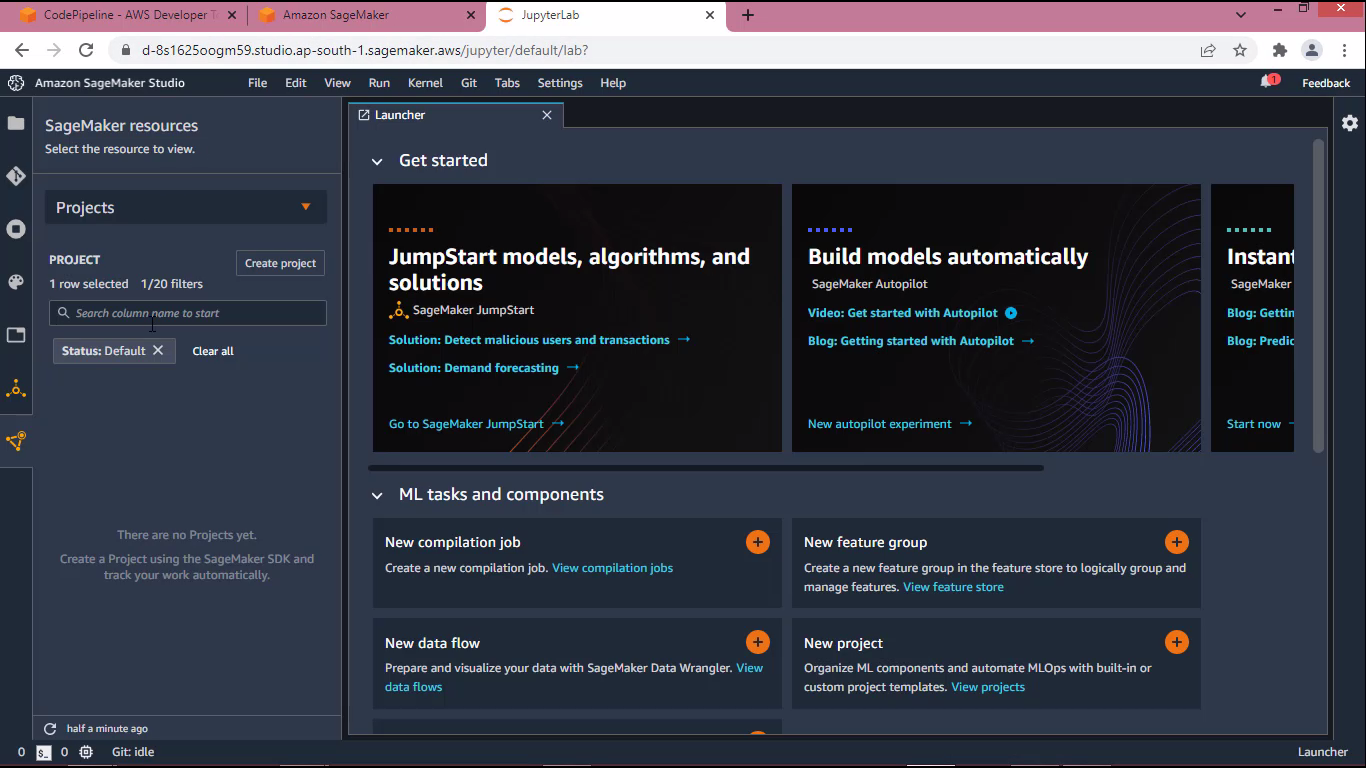
Using SageMaker project Data scientists and Developers can work on machine learning business problems with a SageMaker-provided MLOps template as well as your own custom templates that automates the model building and deployment pipelines using continuous integrations and continuous delivery (CI/CD).

Each SageMaker project has a unique name and ID that are passed to all SageMaker and AWS resources created in the project. Following are the list of entities contained in the project:

* Data gathering
* Data analysis
* Data transformation/preparation
* Model training & development
* Model validation
* Model serving
* Model monitoring
* Model re-training.

Following are the steps in SageMaker project with an in-built template:

* An in-built template means one or more repositories with sample code for building and deploying ML solutions. We will clone these repositories locally to explore the code provided by SageMaker and modify it according to our needs. Own this code and we can use the repositories as version control for our work.
* AWS SageMaker pipeline gives steps for data preparation, training, model evaluation, and model deployment. AWS CodePipeline or Jenkins pipeline (third-party open source) that runs our SageMaker pipeline every time we check in a new version of the code.
* A model group that contains model versions. Each time the SageMaker pipeline runs, and the resulting model version is accepted manually in the conditional validation step, a new model version is deployed to a SageMaker endpoint.
* **Screenshots**



## Fig 1 - Getting started with Sagemaker studio interface

## 

## Fig 2 - Creating the project

## 

## Fig 3 - Cloning the git repository locally

## 

## Fig 4 - Pipeline execution starts

## 

## Fig 5 - Manually Updating the model by Dev team

## 

## Fig 6 - Manually updating the model by Operation team

## 

## Fig 7 - Committing the changes in Git

## 

## Fig 8 - Manually updating the model by Dev team after Git commit

## 

## Fig 9 - Manually updating the model by Operation team after Git commit

## 

## Fig 10 - Model deployed successfully in prod and Endpoint has been generated

* **References:**

1. <https://github.com/aws/amazon-sagemaker-examples/tree/main/sagemaker-pipelines/tabular/customizing_build_train_deploy_project/modelbuild/pipelines/customer_churn>

2. <https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-projects-why.html>

3. <https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-projects-whatis.html>

4.  [https://neptune.ai/blog/mlops](%20https://neptune.ai/blog/mlops)